REMARKS

Claims 1, 2, 5, 9, 18, 19, 22-25 34-36, 39, and 52-68 have been amended. Claims 1-68 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 101 Rejection:

The Office Action rejected claims 52-68 under 35 U.S.C. § 101 as being directed to non-statutory subject matter. Applicants traverse this rejection. However, to expedite prosecution, claims 52-68 have been amended to recite a *computer-accessible storage medium comprising program instructions, wherein the program instructions are computer-executable to implement*. Applicants respectfully request removal of the § 101 rejection of claims 52-68.

Section 103(a) Rejection:

The Office Action rejected claims 1-8, 10-13, 16, 34-59, 61-64 and 67 under 35 U.S.C. § 103(a) as being unpatentable over Nagarajayya (U.S. Patent 6,125,402) in view of Venners ("Inside the Java Virtual Machine"). Applicants respectfully traverse this rejection for at least the following reasons.

Regarding claim 1, contrary to the Examiner's assertion, the cited references fail to disclose, alone or in combination, program instructions executable by a processor to implement *a remote class loader mechanism configured to detect [a generated] indication that a class needed to execute code has failed to be loaded by a default class loader from one or more local locations indicated by a class path of the default class loader, obtain the class from a remote system via a network, and store the class in a location indicated by the class path of the default class loader on the system; wherein the default class loader is configured to load the class from the location indicated by the class path*.

Page 1, line 10-page 3, line 5 of the Background section of the instant application generally describes the dynamic loading of classes using class loaders in the prior art. In particular, page 3, lines 4-5 describes that:

> In Java, to use a class, the class has to be in the class path of the default class loader, or alternatively a **custom** class loader may be provided.

The Venners reference is an overview of the Java Virtual Machine (JVM). Chapters 1-4 "give a broad overview of Java's architecture". A careful review of the Venners reference, particularly an overview given in Chapter 3, page 2, shows that Venners' description of Java's method of loading classes using class loaders is consistent with what is described in the Background section of the instant application. For example, Venners, in Chapter 3, page 2, gives the following description of how Java's class loading mechanism works:

> Imagine that during the course of running the Java application, a request is made of your [custom] class loader to load a class named Volcano. Your [custom] class loader would first ask its parent, the **class path class loader**, to find and load the class. The **class path class loader**, in turn, would make the same request of its parent, the installed extensions class loader. This class loader, would also first delegate the request to its parent, the bootstrap class loader. Assuming that class Volcano is not a part of the Java API, an installed extension, or **on the class path**, all of these class loaders would return without supplying a loaded class named Volcano. When the **class path class loader** responds that neither it nor any of its parents can load the class [i.e., the requested class is **not on a class path**], your [custom] class loader could then attempt to load the Volcano class in its **custom** manner, by downloading it across the network. Assuming your [custom] class loader was able to download class Volcano, that Volcano class could then play a role in the application's future course of execution.

In contrast to the above description from Venners, claim 1 of the instant application recites that a *default class loader* determines that a class needed to execute code on the system is not stored in one or more local locations indicated by the class path (and thus fails to load the class). An indication is generated that the class is not loaded. A remote class loader mechanism then detects the indication. The remote class loader mechanism obtains the class from a remote system via a network and stores the class in a location indicated by the class path. The default class loader (**not** a custom class loader, as in the Venners reference, and as the Java class loading mechanism described by Venners and described in the Background section of the instant application operates) <u>then loads the class from the location indicated by the class path</u>.

To summarize, Venners discloses that, if the "default class loader" (**class path class loader**) cannot locate a class in a location on its class path, the "default class loader" notifies a "custom class loader" associated with the class, which then attempts to load the class, possibly from a remote location. In contrast, claim 1 of the instant application recites that, if the "default class loader" cannot locate a class in a location on its class path, a "remote class loader mechanism" locates the class on a remote system and stores the class in a location indicated by the default class loader's class path. The <u>default class loader</u> then loads the class from the location.

Thus, what claim 1 of the instant application recites is <u>clearly distinct</u> from the Java class loading mechanism as described by the Venners reference.

The Nagarajayya reference discloses a method and system for executing one of several forms of a multi-purpose program. If the program is invoked for executing in an applet form, Nagarajayya discloses that the system makes dynamic calls to a remotely accessible file library for operation of a specific function of the program. (Nagarajayya, Abstract). In col. 9, lines 27-42, Nagarajayya discloses that classes can be "dynamically loaded" from class libraries on a Web server:

> In step 340, the program determines if the applet needs other classes to run that are not available in the subset of classes downloaded in step 330…If more classes are needed by the applet, they are dynamically loaded from the class libraries on the Web server and executed on the client machine as shown in step 345.

In col. 7, line 63-col. 8, line 4, Nagarajayya discloses that a "described embodiment of the present invention" uses a Java-enabled browser that allows it to run Nagarajayya's "applet programs":

> First client machine 202 is running a Java.TM.-enabled browser 234 which has downloaded a Web site 236 as defined by HTML document 224

> and frame 214. In the described embodiment browser 234 is the Hot Java.TM. browser available from Sun Microsystems of Mountain View, Calif. Frame 214 is downloaded onto client machine 202, allowing it to run applet programs, as formed by integrated program 218 and classes 228, as needed based on the user's actions in Web site 236.

The Nagarajayya reference nowhere teaches or suggests that Nagarajayya's system uses anything other than Java's class loader mechanism as described in the Venners reference and in the Background section of the instant application to dynamically load classes from the class libraries on the Web server. In other words, for one of Nagarajayya's applets to use a class, the class <u>has to be locally located in the class path of the default class loader, or alternatively a **custom class loader** has to be provided to load the class from a remote location.</u> Nagarajayya does not teach or suggest that, if the "default class loader" cannot locate a class in a location on its class path, a "remote class loader mechanism" locates the class on a remote system and stores the class in a location indicated by the default class loader's class path, and that the <u>default class loader</u> then loads the class from the location.

Applicants remind the Examiner that, to establish a *prima facie* case of obviousness of a claimed invention, <u>all claim limitations must be taught or suggested by the prior art</u>. *In re Royka*, 490 F.2d 981, 180 U.S.P.Q. 580 (C.C.P.A. 1974), MPEP 2143.03. As shown above, the cited art does not teach or suggest all limitations of claim 1. For example, the cited references, alone or in combination, do not teach or suggest that, if the "default class loader" cannot locate a class in a location on its class path, a "remote class loader mechanism" locates the class on a remote system and stores the class in a location indicated by the default class loader's class path, and that the <u>default class loader</u> then loads the class from the location.

Furthermore, the Examiner's stated motivation for combining the references, "to understand how a Java applet load and utilize classes that are needed at runtime" is not a motivation for combining the references. It is if anything simply a "motive" for understanding <u>how the Nagarajayya system already works</u>, since the Nagarayjayya system already employs the Java class loading method described in Venners. Moreover, the Examiner's stated motivation is merely conclusory.

Furthermore, even if the Nagarajayya and Venners references were combined, the combination would not produce anything like was is recited in claim 1 of the present application. It would simply produce a system that handles the loading of classes using the Java class loading method as disclosed in Venner, which it is clear that Nagarajayya's system already uses. In other words, it would simply produce the Nagarajayya system as disclosed in the Nagarajayya reference. **Since the Nagarajayya reference already employs the class loading mechanism disclosed in Venners, there is and can be no motivation to combine the references in any case.**

Thus, for at least the reasons presented above, the rejection of claim 1 is not supported by the cited prior art and removal thereof is respectfully requested. Similar remarks as those above regarding claim 1 also apply to claim 34, 35, and 52.

The Office Action rejected claims 9, 14, 15, 17-33, 60, 65, 66 and 68 as being unpatentable over Nagarajayya in view of Venners, and further in view of Babaoglu, et al. ("Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems") (hereinafter "Babaolgu"). Applicants respectfully traverse this rejection for at least the reasons given above in regards to claim 1.

In further regard to claim 17, contrary to the Examiner's assertion, the cited references fail to disclose, alone or in combination, program instructions executable by a processor to implement *wherein the system and the remote system are configured to participate in a distributed computing system on the network for submitting computational tasks in a distributed heterogeneous networked environment that utilizes peer groups to decentralize task dispatching and post-processing functions and enables a plurality of jobs to be managed and run simultaneously.* The Examiner simply asserts that Babaoglu teaches "peer-to-peer application can be implemented in Java (page 7, section 4)" and that "it would have been obvious…to apply the teachings of Babaoglu to the system of Nagarajayya because it presents a framework supporting a new approach for building P2P application in which resource can be sharing by direct exchange between peer nodes."

The Babaoglu reference describes:

> …Anthill, a framework to support the design, implementation and evaluation of P2P applications based on ideas such as multi-agent and evolutionary programming borrowed from CAS. An Anthill system consists of a dynamic network of peer nodes; societies of adaptive agents travel through this network, interacting with nodes and cooperating with other agents in order to solve complex problems. (Babaoglu, Abstract).

However, the Examiner has simply asserted that "Babaoglu teaches "peer-to-peer application can be implemented in Java", and has not provided any argument or reference that Babaoglu teaches or suggests <u>all</u> of the claim limitations recited in claim 17 of the instant application. For example, the Examiner has provided no argument or reference, nor can the Applicants find anything, from the Babaoglu reference that teaches or suggests that the Babaoglu "Anthill" framework is a <u>distributed computing system for submitting computational tasks in a distributed heterogeneous networked environment</u> or that the Babaoglu "Anthill" framework *<u>decentralizes task dispatching and post-processing functions</u>* and *<u>enables a plurality of jobs to be managed and run simultaneously</u>*.

Applicants remind the Examiner that, to establish a *prima facie* case of obviousness of a claimed invention, <u>all claim limitations must be taught or suggested by the prior art</u>. *In re Royka*, 490 F.2d 981, 180 U.S.P.Q. 580 (C.C.P.A. 1974), MPEP 2143.03. As shown above, the cited art does not teach or suggest all limitations of claim 1.

Furthermore, the Examiner's stated motivation for combining the references, "because it presents a framework supporting a new approach for building P2P application in which resource can be sharing by direct exchange between peer nodes", is not a motivation that is directly relevant to Nagarajayya's method and system for <u>executing a</u>

particular form of a multi-purpose program in a distributed computing environment. It is **not at all** "obvious" how Nagarajayya's disclosed method and system would benefit from Babaoglu "Anthill" framework, nor is it obvious as to how Babaoglu's "Anthill" framework could, or even if it could, be implemented in Nagarajayya's method and system. **Furthermore, the stated motivation is not a motivation that is directly relevant to the functioning or application of what is recited in claim 1 of the instant application.** Moreover, the Examiner's stated motivation is merely conclusory.

Furthermore, even if the Nagarajayya and Babaoglu references were combined, the combination would not produce anything like was is recited in claim 1 of the present application.

Thus, for at least the reasons presented above, the rejection of claim 17 is not supported by the cited prior art and removal thereof is respectfully requested. Similar remarks as those above regarding claim 1 also apply to claim 51 and 68.

Regarding independent claim 18, Applicants respectfully traverse this rejection for at least the reasons given above in regards to claim 1 and claim 17.

Applicants also assert that the rejection of numerous ones of the dependent claims is further unsupported by the cited art. However, since the rejection has been shown to be unsupported for the independent claims, a further discussion of the dependent claims is not necessary at this time.

CONCLUSION

Applicants submit the application is in condition for allowance, and prompt notice to that effect is respectfully requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5681-65900/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicant(s)

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: April 16, 2007